基于纯Python的SKA-RASCIL中的天线增益校准实现与优化

黎静[1] 曹忠[1] 王锋[11] 邓辉[1] 梅盈[1] 戴伟[2]

1、广州大学物理与材料科学学院/天体物理中心，广州,510006

2. 昆明理工大学云南省计算机技术应用重点实验室，昆明，650051

摘要：天线增益是用来衡量天线在指定方向接收和发射信号能力的参数。随着平方公里阵列（Square Kilometre Array，SKA）建设的日趋临近，基于Python语言的射电天文仿真、校准以及成像软件库(Radio Astronomy Simulation, Calibration and Imaging Library ，RASCIL)正在开发完善中，天线增益校准算法的高性能实现是其中重要的组成部分。本文基于RASCIL antsol天线增益校准算法开发工作，首先讨论了antsol算法的基本原理，随后分析说明了基于纯Python的实现过程，在此基础上进一步讨论了Python语言中算法的优化方法。结果表明，通过底层优化后的antsol算法实现可以达到非常高的性能。经过优化的算法将运用到SKA的增益数据校准实验中，本文所介绍的优化方法对其它天文软件的开发有一定的参考价值。

关键词：RASCIL；antsol；天线增益；算法优化

中图分类号：TP274

一、前言

宇宙再电离、脉冲星、寻找外星智慧生命体等科学研究是当前天文领域的研究热点，而要实现这些研究目标，就迫切需要射电望远镜具有更高的灵敏度和分辨率。凭借其巨大的信号接收面积和千公里的基线长度，新一代射电望远镜—平方公里阵（SKA）具有超高灵敏度、超高分辨率的特点。同时，SKA将以纳秒级的采样频率工作，这意味着SKA将产生无比庞大的观测数据量。由于结合了高分辨率、高动态范围、大数据等一系列最新的射电天文技术，SKA将给天文学研究带来革命性的影响。高质量成像是实现SKA科学目标的关键，高精度的校准是实现大视场、高动态范围成像的前提，是将原始观测数据转变为科研级数据的必经过程。项目组前期在SKA基础数据格式MS文件的生成[1]、宽视场成像算法分析方面有了较为深入的研究[2]。针对极为微弱且错综复杂的各类系统与环境效应的校准是困扰当前SKA科学研究的关键因素，为了进一步提高成像精度，天线增益的校准成为亟待深入研究的工作。射电干涉阵对目标源进行观测获得可见度数据，对这些可见度数据进行傅里叶逆变换，可以得到天空亮度分布图[3]。可见度数据由天线阵的相关器输出，但实际上电离层的变化、温度变化、地面接收、天线阻塞、各种电子元件的噪声接收、背景温度等造成天线的相位误差和幅度误差[4]，导致天线增益发生变化，对相关器的输出数据产生干扰，严重影响可见度数据的真实性。因此，为了获得尽可能真实

---

[1] 基金项目：国家重点研发计划（2018YFA0404603），国家自然科学基金委员会-中国科学院天文联合基金资助项目(U1831204,U1931141)，国家自然科学基金国际合作项目（11961141001），云南省重点研发计划(2018IA054)，国家自然科学基金青年科学基金资助项目（11903009）资助。

黎静，女，硕士研究生，研究方向：天文技术与方法.Email：lijing@cnlab.net

王锋，男，教授，研究方向：天文技术与方法，Email：fengwang@gzhu.edu.cn

的天空亮度分布图以进行下一步的天文研究，必须对射电干涉阵的观测数据进行天线增益校准。

目前，天线增益校准的方法有标校准法、迭代自校准法和射电源法。在成像应用中，常用的校准技术有两种。一种是“盲”迭代自校准技术，将参数从初始预估值开始进行调整，直到图像结果与事先建立的参数模型（通常是一个点源模型）匹配为止。另一种是在天空中选取一个已知的、相对稳定的强点源进行专门的校准观测[5]。

在过去的几十年中，针对射电干涉阵成像存在相位误差和幅度误差的问题，科学家们给予了相当大的关注，并在增益校准算法上取得了一定的成功[6]。1974年，Rogers[7]等人首次把闭合相位运用在甚长基线干涉测量(Very Long Baseline Interferometry ，VLBI)的相位校准中。后来Fort和Yee[8]、Readhead和 Wilkinson[9]先后在Rogers的基础上进一步发展增益校准算法。但是这些算法有比较大的局限性，只对望远镜数量比较少的射电干涉阵的数据有效，且要求不同基线的信噪比相同。因此Schwab[10]提出了更普遍适用的算法。随后，Cornwell和Wilkinson[11]以及Thompson等[3]也改进和完善了增益校准算法，并分别应用在多望远镜射电连接干涉仪（Multi Telescope Radio Linked Interferometer，MTRLI）和甚大阵（Very Large Array ，VLA）的图像数据处理中。由于性能良好，存在很多年的antsol算法至今仍在GMRT（Giant Metrewave Radio Telescope）和CASA（Common Astronomy Software Applications）使用，是AIPS的默认算法。除了经典“antsol”算法外，还有一种叫“Stefcal”的算法[12]，该算法在低频阵列（Low Frequency Array ，LOFAR）校准管线的几个阶段中得到了实现。

SKA的天线增益校准是最终SKA获得高动态范围成像的关键工作之一，在基于Python语言的射电天文仿真、校准以及成像软件库(RASCIL)的开发完善过程中，天线增益校准的算法实现是其中的一个重要模块。本文对利用Python语言实现天线增益校准算法开发工作进行了研究，重点分析讨论了代码实现与优化问题。

二、天线增益校准算法（antsol）基本原理

在综合孔径成像中，不同天线接收到的信号通过乘法电路和时间平均电路进行组合以及输出。乘法电路和时间平均电路组成相关器。干涉阵的每个相关器的输出为天线信号和系统噪声的总和。对于天线对p和q，相关器的输出为：

$$\rho_{pq}^{obs} = g_p g_q^* \rho_{pq}^o + \varepsilon_{pq} \quad (1)$$

其中，ρ_{pq}^{obs}表示由天线p、q组成的基线观测到的可见度，ρ_{pq}^o表示模型可见度模型（通常是一个点源模型），ε_{pq}表示由天线p和天线q组成的基线的叠加噪声，g_p和g_q^*分别为天线p的复增益因子、天线q的复增益因子的共轭复数。对于任意天线k，增益可写成：

$$g_k = |g_k| e^{-i\varphi k}$$

（2）

其中$|g_k|$表示天线接收信号的幅值，φ表示信号基于天线的相位。因此，p、q

天线对的复增益为：

$$G_{pq} = g_p g_q^*$$

（3）

给定 ρ_{pq}^{obs} 并且已知模型可见度 ρ_{pq}^{o} ，就可以求解天线复增益 G_{pq} 。假设天线复增益是独立的，且符合高斯概率密度函数（这意味着实部和虚部是独立的高斯随机过程），则可以通过使给定的函数S最小来估计 g_p 。

$$S=\sum_{\substack{p,q\\p\neq q}}\left|G_{pq} - g_p g_q^*\right|^2 w_{pq}$$

（4）

其中， w_{pq} 表示**权重值**。将S分别对复增益的实部 g_p^R 、虚部 g_p^I 求偏导数并令其为零，合并实部、虚部：

$$g_p = \frac{\sum_{\substack{q\\q\neq p}} G_{pq} g_q}{\sum_{\substack{q\\q\neq p}} \left|g_q\right|^2}$$

（5）

在校准完成后，可以估算残差 r_{pq}，为了使得残差尽可能小，一般在校准时都采用多次迭代的方法来提高校准的精度。

$$r_{pq} = G_{pq} - g_p g_q^*$$

（6）

三、算法实现

如前所述，antsol算法已经有多种实现方式，包括Fortran版本与C版本等，从编程实现角度来看并不困难。为了保持与RASCIL的目标一致，全部程序采用纯Python开发，同时软件包尽可能为Python的常用软件包。天线增益校准的程序名保存在rascil软件包的processing_components/calibration下，程序文件名为solver.py，所有的源程序均可以gitlab上下载。

SKA的软件设计规范要求，在桥接阶段的代码应采用Python语言开发。为了实现分布式计算，SKA-SDHP的代码均采用DASK(https://www.dask.org)作为多任务分布计算框架。Dask 库可以将计算扩展到多个内核甚至多个机器。因此在天线增益校准开发中只实现相应的模块，在模块内不再使用多任务并行计算。同时，为确保代码的可读性，目前阶段不考虑GPU并行。为此，在开发RASCIL相应模块的过程中，为了加快进度和确保结果正确，我们直接以最早的美国甚大阵的源程序进行了改写，这一源程序采用的基础算法来自于Thomspon这本经典书籍的附录1[3]。

结合SKA数据处理需要，在实现中综合考虑了天线增益校准主的两个过程，一个是计算天线增益（见公式5），第二是计算残差（见公式6）。算法实现的相关函数说明见表1。

表1. 算法实现的相关函数说明
Table 1. The Implementation of the antsol algorithm

No.	Function Name	Desctiption
1	solve_gaintable	Solve a gain table by fitting an observed visibility to a model visibility
2	solve_antenna_gains_itsubs_scalar	Solve for the antenna gains, x(antenna2, antenna1) = gain(antenna1) conj(gain(antenna2))
3	solve_antenna_gains_itsubs_matrix	Solve for the antenna gains using full matrix expressions
4	solve_antenna_gains_itsubs_vector	Solve for the antenna gains using full vector expressions
5	gain_substitution_matrix	Gain calculation subfunction(matrix expressions), called in function 2.
6	gain_substitution_scalar	Gain calculation subfunction(scalar expressions), called in function 3.
7	gain_substitution_vector	Gain calculation subfunction(vector expressions), called in function 4
8	Solution_residual_matrix	Calculate residual across all baselines of gain for point source equivalent visibilities, the results are expressed in matrix form, called in function 5
9	Solution_residual_vector	Calculate residual across all baselines of gain for point source equivalent visibilities, the results are expressed in vector form, called in function 7

受限于篇幅，我们以一个典型的solution_resudial_matrix函数的实现为例，原始代码（略）是标准的Fortran代码，因此简单地把循环、赋值语句按Python语法进行修改。函数的几个参数：gain表示计算出的增益数组，x表示点源等效可见度数组，xwt表示点源等效权重数组，函数返回的是残值数组f，具体代码见表2。

表2. solution_resudial_matrix函数的代码
Table 2. Code for solution_resudial_matrix function

```
def solution_residual_matrix(gain, x, xwt):
    nants, _, nchan, nrec, _ = x.shape

```

```
4       residual = numpy.zeros([nchan, nrec, nrec])
5       sumwt = numpy.zeros([nchan, nrec, nrec])
6
7       # 以下代码开始残值的计算
8       for ant1 in range(nants):   #第一层循环：天线
9           for ant2 in range(nants): #第二层循环：天线
10              for chan in range(nchan): #第三层循环：通道
11                  for rec1 in range(nrec):
12                      for rec2 in range(nrec):
13                          error = x[ant2, ant1, chan, rec2, rec1] - \
14                                  gain[ant1, chan, rec2, rec1] * numpy.con
15  jugate(gain[ant2, chan, rec2, rec1])
16                          residual[chan, rec2, rec1] += (error * xwt[ant2,
    ant1, chan, rec2, rec1] * numpy.conjugate(error)).real
17                          sumwt[chan, rec2, rec1] += xwt[ant2, ant1, chan,
    rec2, rec1]
18
19
20      residual[sumwt > 0.0] = numpy.sqrt(residual[sumwt > 0.0] / sumwt[sum
21  wt > 0.0])
22      residual[sumwt <= 0.0] = 0.0
        return residual
```

以上代码段中，从第8行起五重循环来实现不同天线和不同通道的处理，用numpy.conjugate函数实现共轭函数。第13行计算误差值，第16行计算残差，这样的代码风格与Fortran或C语言程序一致，程序易于理解。在开发过程中，这样的代码也确保了程序的快捷移植与实现，结果也符合预期。

但随之而来的问题是在测试中发现这些代码运行效率非常低下。通过PyCharm开发环境中的Profile功能以及一系列的实验，我们对代码性能的瓶颈进行了分析，本质是因为Python是解释型语言。在编译型语言中，多重循环的方式不会影响整体性能。但是Python作为解释型脚本，这样的实现模式效率非常低下。代码13行到18行通过循环计算不同天线和不同通道数据的误差和残差累加和，这在Profile过程中是耗时最大的代码。

此外，虽然Python中最有特色的花式索引技术在实践中利于编程，如代码19与20行，这在传统的C/C++或Fortran编程中一般是通过循环加条件判断来实现，但性能测试表明这种花式索引技术在Python下效率也不高，这一部分是整个函数第二耗时的地方。

四、代码优化方法

Python程序代码性能优化的方法有很多，但SKA项目相关软件规定采用纯Python开发，并且要求尽量使用Python常用的软件包，而不建议采用过多的第三方软件包。这限制了利用C/C++开发再用Python来调用。因此，针对以上性能低下的问题，进一步研究了Python代码的优化以及性能调优问题。通过对上述代码进行性能分析(profile)，主要的性能局限在两个方面，一是用代码实现的矩阵计算，二是Python中的numpy所用的花式索引。

4.1 矩阵计算调优

仔细分析上述代码，结合公式（5）和（6），算法实现的核心需求就是实现一个天线和对应天线的共轭矩阵数据相乘。因此要提高其性能，满足SKA桥接阶段数据阶段的要求，就需要充分利用Python的语言特性。

Numpy是Python语言中的重要软件包，其中提供了大量的矩阵操作函数。Numpy采用C/C++语言开发，因此在实际处理中理论上性能会远优于纯脚本代码。Numpy软件包中提供了一系列的数组或矩阵处理函数，包括数组（矩阵）的内积、外积等。我们依次对Numpy所提供的数组计算函数进行了性能测试，结果表明Numpy软件包自带的这些函数效率都远超过Python脚本实现的效率。显然，优化的关键在于能够一次实现在第三章出现的多天线、多通道的数据处理，尽可能避免Python中的循环。

通过尝试，最终我们使用了numpy中的einsum函数。所谓的einsum ，全称为Einstein summation convention（爱因斯坦求和约定），又称为爱因斯坦标记法，是爱因斯坦 1916 年提出的一种标记约定。简单来说，应用 einsum就是省去求和式中的求和符号。einsum函数的最大优势是，可以把几个多维数据组中的某几个维度进行乘积或者乘积求和。

对于射电干涉阵数据处理来说，一般存在天线（或者基线）、频率通道、极化等多个维度，用einsum可以根据需要灵活地进行矩阵计算，如矩阵转置、矩阵乘法、求迹、张量乘法、数组求和等等，如果分别用transopse、sum、trace、tensordot等函数实现，不但复杂，还容易出错。此外，由于开发团队已经对einsum进行过多次优化，整体来看einsum自身性能较好，是一个非常有效的方法。但einsum开发的难度在于开发者需要充分了解其计算的原理，对数组中的数据处理过程要有非常清楚的认识。

4.2 花式索引的优化方法

对一个数组的部分数据进行处理是编程中常见的要求，在C/C++或Fortran中，一个循环语句就可以实现对数组中部分数据的处理，并保持较高的效率。但Python中，由于脚本解释的关系，循环语句效率不高。通过对相关函数的测试，最终我们发现，putmask函数是当前Numpy软件包中实现数组部分元素替换的最快函数。

4.3 最终实现与性能对比

经过一系列改进，在RASCIL中最终实现的代码如表3所示。

表 3. 优化后的solution_residual_matrix函数的代码

Table 3. Optimized code for the solution_residual_matrix function

```
def solution_residual_matrix(gain, x, xwt):
    nants, _, nchan, nrec, _ = x.shape
    n_residual = numpy.zeros([nchan, nrec, nrec])
    n_sumwt = numpy.zeros([nchan, nrec, nrec])

    n_gain = numpy.einsum('i...,j...->ij...', numpy.conjugate(gain), gain)
    n_error = numpy.conjugate(x - n_gain)
    nn_residual = (n_error * xwt * numpy.conjugate(n_error)).real
    n_residual = numpy.einsum('ijk...->k...', nn_residual)
```

```
11      n_sumwt = numpy.einsum('ijk...->k...', xwt)
12
13      numpy.putmask(n_residual,n_sumwt > 0.0, numpy.sqrt(n_residual[n_sum
14  wt > 0.0] / n_sumwt[n_sumwt > 0.0])
15      numpy.putmask(n_residual, n_sumwt <= 0.0,0.0)
        return n_residual
```

程序的第6行，通过einsum计算出一个天线和天线的共轭矩阵相乘，然后通过简单的求解就可以获得误差（见第7行）。其中最有代表性的是第6，9和10行，直接将各个天线的可见度数据一次处理完。整个功能实现与表2的完全一致。同时，程序13与14行采用putmask函数来替换之前的花式索引。

五、性能分析与讨论

经过多次优化和完善代码，优化后的程序计算性能得到了明显的提高。我们在服务器（CentOS 7.8操作系统，Intel E5 2620 V4 CPU和128G内存）的环境中，完成仿真了SKA1-LOW的可见度函数，进而利用可见度函数对计算耗时进行了测试。

表4列出了优化前后的计算耗时对比，优化前耗时即为原始代码的profile结果，优化后耗时即为对矩阵计算和花式索引进行优化后的profile结果。其中gain_substitution_matrix函数加速比大约达到100、Solution_residual_matrix函数加速比约为250、Solution_residual_vector加速比约为500。优化后的代码满足了SKA桥接阶段的性能要求。通过优化前后代码的性能对比，我们对Python语言天文软件的开发有了更多的理解。

（1）不要依赖传统的C/C++或Fortran的编程基础来开发Python代码，而是从C-Style或者Fortran-Style编程转变到真正的Python-Style。

（2）要深入分析代码的实现模式，尽可能应用Numpy中的内置函数。同时Numpy不同函数的性能有差异，在对性能有需求的情况下需要事先对不同函数进行测试。

（3）Numpy是纯C/C++开发，因为在理想情况下，Python开发的代码也会与C/C++实现代码基本性能相当。由于受到SKA目前阶段代码开发规范的限制，本文没有讨论直接用C/C++实现，或者使用Numba进行实时编译。但本文的结果表明，经过优化的Python代码也能达到与C/C++相当的性能，既满足了性能的要求，又保持了程序代码的可读性。

（4）本文没有将Python实现的代码与传统的Fortran或C/C++实现的代码进行对比，因为在这个方面涉及非常多的相关研究工作。一般来说，C/C++的代码会比Python程序有几十甚至上百倍的效率提升。但从本文的优化效果来看，优化后的计算耗时也与纯C/C++代码基本相当。

表4. 优化前后计算耗时对比
Table 4. The time consumption comparison between and after optimization

No.	Function Name	Time consumption before optimization (ms)	Time consumption after optimization (ms)
1	gain_substitution_matrix	12630	121
2	gain_substitution_scalar	947	66
3	gain_substitution_vector	18083	489

4	Solution_residual_matrix	5940	24
5	Solution_residual_vector	11433	23

六、结论

本文首先分析了天线增益校准antsol算法的基本原理以及标准C/C++或Fortran代码的移植过程，然后针对Python代码中的算法实现提出了优化方法。最后对优化前后的算法耗时进行试验和比较。代码实现结果表明，通过底层优化后的Python算法可以达到与传统Fortran语言实现的antsol算法相近的性能。经过优化的算法将运用到SKA的增益数据实验中，本文所介绍的优化方法对其它天文软件的开发有一定的参考价值。

参考文献

[1] 孙浩民,邓辉,梅盈等.基于Python-casacore的射电测量集文件生成方法[J].天文研究与技术,2020,17(02):210-216.

[2] 于晓雨,邓辉,梅盈等.宽视场成像网格化算法中w-plane最优经验值研究[J].天文研究与技术,2019,16(02):218-224.

[3] THOMPSON A, D'ADDARIO L. Frequency response of a synthesis array: Performance limitations and design tolerances [J]. Radio Science, 1982, 17(02): 357-69.

[4] BHATNAGAR S: Tech. rep., National Centre for Radio Astrophysics, Pune, 1999.

[5] BOONSTRA A-J, VAN DER VEEN A-J. Gain calibration methods for radio telescope arrays [J]. IEEE Transactions on Signal Processing, 2003, 51(1): 25-38.

[6] WIERINGA M H. An investigation of the telescope based calibration methods ‘redundancy’and ‘self-cal’ [J]. Experimental Astronomy, 1992, 2(4): 203-25.

[7] ROGERS A, HINTEREGGER H, WHITNEY A, et al. The structure of radio sources 3C 273B and 3C 84 deduced from the'closure'phases and visibility amplitudes observed with three-element interferometers [J]. The Astrophysical Journal, 1974, 193(293-301.

[8] FORT D, YEE H. A Method of Obtaining Brightness Distributions from Long Baseline Inter-ferometry [J]. Astronomy and Astrophysics, 1976, 50(19.

[9] READHEAD A, WILKINSON P. The mapping of compact radio sources from VLBI data [J]. The Astrophysical Journal, 1978, 223(25-36.

[10] SCHWAB F. Robust solution for antenna gains [J]. Very Large Array (VLA) Scientific Memorandum, 1982, 136(1-20.

[11] CORNWELL T, WILKINSON P. A new method for making maps with unstable radio interferometers [J]. Monthly Notices of the Royal Astronomical Society, 1981, 196(4): 1067-86.

[12] SALVINI S, WIJNHOLDS S J. StEFCal—An Alternating Direction Implicit method for fast full polarization array calibration; proceedings of the 2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS), F, 2014 [C]. IEEE.

The Implementation and Optimization of Antenna Gain Calibration Based on Pure Python for SKA-RASCIL

Li Jing[1], Cao Zhong[1], Wang Feng[11], Deng Hui[1], Mei Ying[1], Dai Wei[2]

1. Center for Astrophscis, School of Physics and Materials Science, Guangzhou University, Guangzhou, 510006, China.

2. Key Laboratory of Applications of Computer Technology of the Yunnan Province，Ku

nming University of Science and Technology， Kunming， 650051，China，

Abstract: Antenna gain is a parameter that measures the ability of an antenna to receive and transmit signals in a specified direction. With the construction of the Square Kilometre Array (SKA), the Python-based radio astronomy simulation, calibration and imaging software library (RASCIL) is being developed and improved, and the high-performance implementation of the algorithm in antenna gain calibration is an important part of it. Based on the development of the RASCIL antsol antenna gain calibration algorithm undertaken, this study first discusses the basic principles of the antsol algorithm, and then analyzes the implementation process based on pure Python scripts, and further discusses the optimization method of the algorithm in the Python language on this basis. The result shows that the implementation of the optimized antsol algorithm can achieve very high performance. The optimized algorithm will be used in the gain data calibration experiment of SKA. The optimization method introduced in this paper has certain reference value for the development of other astronomical softwares.
Key words: RASCIL; Antsol; Antenna gain; Algorithm optimization